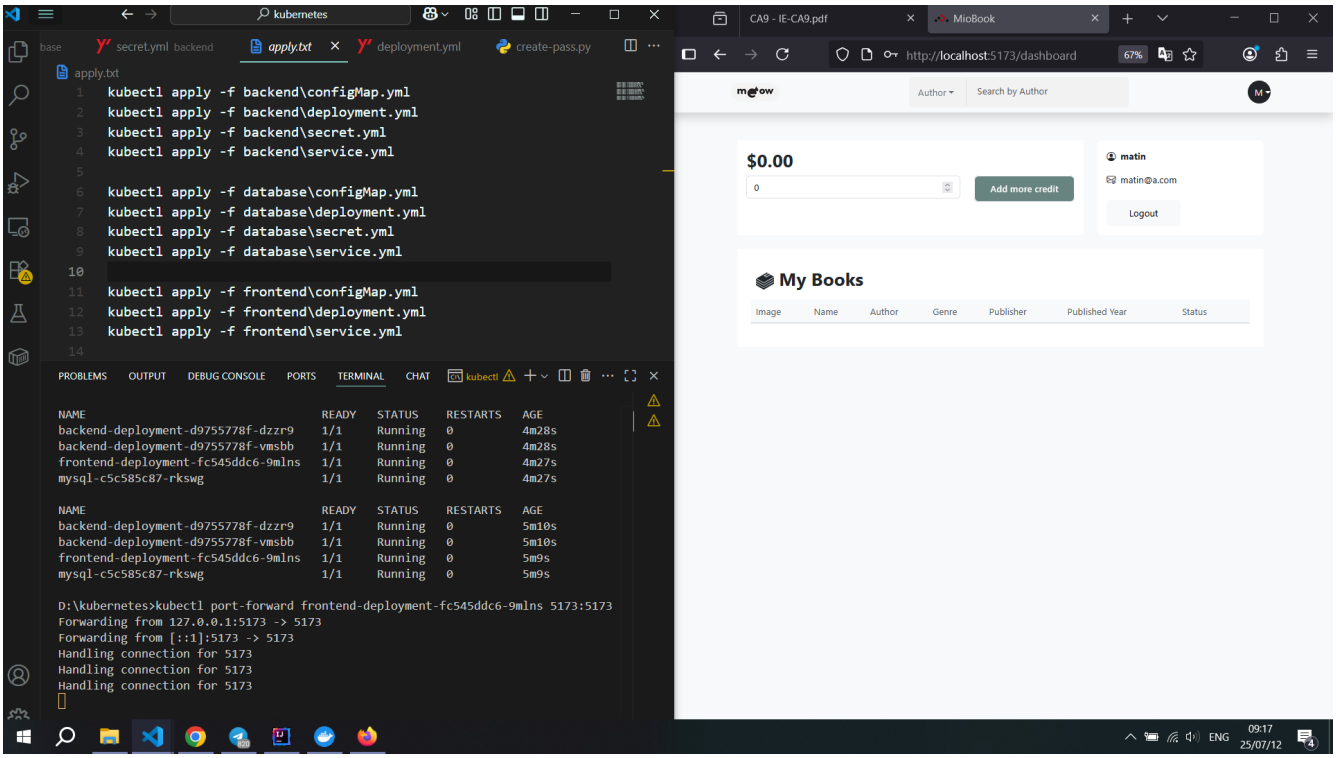
مهندسی اینترنت

متین بذرافشان-نرگس الیاسی

-۸۱۰۱۰۰۰۹۳

۸۱۰۱۰۰۲۵۸

پروژه نهم



سوال اول

انواع مختلفی از Service وجود دارند که هرکدام کاربردها و محدودیت‌های خاص خود را دارند:

- **ClusterIP**

این نوع Service به صورت پیش‌فرض یک آدرس IP داخلی به سرویس اختصاص می‌دهد که فقط از داخل کلاستر قابل دسترسی است. در واقع، این سرویس برای ارتباط داخلی بین پادها و سایر منابع داخل کلاستر استفاده می‌شود و امکان دسترسی از بیرون کلاستر وجود ندارد.

- **NodePort**

این Service به هر نود کلاستر یک پورت مشخص اختصاص می‌دهد که از طریق آن می‌توان از خارج کلاستر به سرویس دسترسی داشت. محدوده پورت‌های مجاز معمولاً بین ۳۰۲۴ تا ۳۲۷۶۷ است. این روش ساده و سریع است، اما ممکن است محدودیت‌هایی از نظر امنیت و مدیریت ترافیک داشته باشد.

- **LoadBalancer**

این نوع Service در محیط‌های ابری کاربرد دارد و یک Balancer Load خارجی برای سرویس ایجاد می‌کند که ترافیک را به صورت هوشمند بین پادها تقسیم می‌کند. استفاده از این Service وابسته به پشتیبانی Provider Cloud است و ممکن است هزینه‌های اضافی به همراه داشته باشد.

- **ExternalName**

این Service نوعی ریدایرکت DNS به یک نام دامنه خارجی است. به این معنا که درخواست‌ها به جای پادهای داخل کلاستر به یک سرویس خارجی هدایت می‌شوند. این روش Balancing Load ندارد و صرفاً برای ارجاع به سرویس‌های خارج از کلاستر مناسب است.

سوال دوم

Namespaceها یک سطح جداسازی در منابع و فضای نام ارائه می‌دهند که باعث می‌شود چندین تیم، پروژه یا اپلیکیشن بتوانند به صورت همزمان روی یک کلاستر مشترک کار کنند بدون اینکه منابعشان با هم تداخل داشته باشد. ایزوله‌سازی منابع: منابعی مثل Pod، Service، Config Map، Secret و غیره در Namespaceها قرار می‌گیرند. این باعث می‌شود که منابع با یک نام مشابه در هایNamespace مختلف بدون برخورد با هم وجود داشته باشند. مثلاً می‌توان دو Pod با نام frontend داشت که در هایNamespace جداگانه هستند.

- مدیریت مجوز ها و دسترسی ها:

با ترکیب Namespace و RBAC (Role-Based Access Control)، می‌توان دسترسی‌ها را به صورت محدودشده برای هر Namespace تعریف کرد. این برای امنیت و مدیریت کاربران اهمیت زیادی دارد.

- سازماندهی و تفکیک تیم‌ها و محیط‌ها:

معمولاً هر تیم یا پروژه یک Namespace اختصاصی دارد. همچنین می‌توان محیط‌های مختلف مثل dev، staging و prod را با Namespaceهای جداگانه از هم تفکیک کرد.

- محدودیت منابع:

می‌توان برای هر Namespace محدودیت منابع (Resource Quota) تعیین کرد، مثلاً حداکثر CPU، حافظه و تعداد پادها، که به کنترل مصرف منابع کمک می‌کند.

مثال از Resource که در Namespace قرار می‌گیرند

- **Pods**
- **Services**
- **Deployments**
- **ConfigMaps**
- **Secrets**
- **NetworkPolicies**

تنظیم Namespace پیش‌فرض با kubectl

```
kubectl scale deployment <deploymentname> --replicas=<number-of-pods> -n <namespace>
```

تنظیم Namespace پیش‌فرض از طریق ویرایش فایل kubeconfig

تغیر فایل kubeconfig که معمولا در /kube/config./ قرار دارد:

```
namespace: my-namespace
```

سوال سوم

در شرایطی که بار (Load) روی سرویس‌های بک‌اند افزایش می‌یابد، نیاز است تعداد پادها به‌صورت پویا افزایش یابد

افزایش دستی تعداد پادها

رای خودکارسازی افزایش یا کاهش تعداد پادها بر اساس مصرف منابع (معمولاً CPU)، از HPA استفاده می‌شود. HPA وضعیت مصرف منابع را بررسی کرده و در صورت عبور از آستانه تعیین‌شده، تعداد پادها را در بازه‌ای مشخص افزایش یا کاهش می‌دهد. بعد از نصب

```
kubectl scale deployment <deployment-name> --replicas=<number> -n <namespace>
```

افزایش خودکار با **HPA**

برای خودکارسازی افزایش یا کاهش تعداد پادها بر اساس مصرف منابع (معمولاً CPU)، از HPA استفاده می‌شود. HPA وضعیت مصرف منابع را بررسی کرده و در صورت عبور از آستانه تعیین‌شده، تعداد پادها را در بازه‌ای مشخص افزایش یا کاهش می‌دهد. بعد از نصب metric-server در کلاستر، در Deployment مربوطه مقادیر resources.requests.cpu مشخص اند. با این دستور HPA ساخته می شود:

```
kubectl autoscale deployment my-backend --cpu-percent=60 --min=3
--max=10 -n backend-namespace
```

یا می توان از فایل YAML برای تعریف آن استفاده کرد. که توسط دستور زیر اعمال خواهد شد:

```
    kubectl apply -f backend-hpa.yaml
```

سوال چهارم

زمانی که تغییراتی روی یک آبجکت مانند Deployment در Kubernetes اعمال می‌شود (برای مثال با اجرای kubectl apply -f یا ویرایش مستقیم YAML)، سیستم تلاش می‌کند وضعیت جاری (Current State) کلاستر را به وضعیت مطلوب (Desired State) جدید که در YAML مشخص شده، نزدیک کند.
این فرآیند توسط چند مؤلفه اصلی در معماری کوبرنتیز مدیریت می‌شود:

- **API Server**

نخست، فایل YAML از طریق kubectl به API Server ارسال می‌شود. API Server رابط اصلی بین کاربران و سایر اجزای داخلی Kubernetes است و تمام درخواست‌ها (مانند ایجاد، به‌روزرسانی یا حذف منابع) ابتدا از این مسیر عبور می‌کنند.

- **etcd** (ذخیره‌ساز وضعیت کلاستر)

API Server اطلاعات جدید را در پایگاه داده مرکزی کلاستر یعنی etcd ذخیره می‌کند. etcd شامل وضعیت کامل و فعلی همه منابع در کلاستر است.

- **Controller Manager**

سپس، Controllerها که در Controller Manager اجرا می‌شوند، تغییرات وضعیت جدید را دریافت می‌کنند. برای مثال، در مورد یک Deployment، کنترلری به نام Deployment Controller مسئول بررسی تفاوت بین وضعیت فعلی (چه تعداد پاد در حال اجرا هستند) و وضعیت مطلوب (تعداد، قالب، و تنظیمات جدید پادها در YAML) است.

این کنترلرها به‌صورت پیوسته وضعیت واقعی سیستم را با وضعیت مطلوب مقایسه کرده و در صورت اختلاف، عملیات لازم برای هماهنگ‌سازی را انجام می‌دهند (مثلاً ایجاد پاد جدید، حذف پاد قدیمی، یا به‌روزرسانی آن‌ها).

- **Scheduler** و **Kubelet**

اگر تغییرات نیاز به ایجاد یا جابه‌جایی پاد داشته باشد، Scheduler تصمیم می‌گیرد که پاد جدید روی کدام نود اجرا شود. سپس Kubelet روی آن نود وظیفه اجرای پاد را به عهده می‌گیرد.

سوال پنجم

Operator در واقع یک کنترلر (Controller) سفارشی است که معمولاً با یک Custom Resource Definition (CRD) همراه می‌شود. این دو با هم اجازه می‌دهند که رفتار یک نرم‌افزار خاص، مانند دیتابیس یا سیستم مانیتورینگ، درون Kubernetes مدیریت شود؛ مشابه همان‌طور که Kubernetes به صورت پیش‌فرض Deployment و Service را مدیریت می‌کند.
مدیریت دستی نرم‌افزارهای stateful در Kubernetes، مخصوصاً در محیط‌های production، پیچیده و مستعد خطاست. استفاده از Operator مزایای زیر را به همراه دارد:

- اتوماسیون کامل فرآیندهای عملیاتی
- افزایش قابلیت اطمینان و تکرارپذیری
- همسویی کامل با مدل اجرایی Kubernetes

- کاهش خطاهای انسانی

سوال ششم

Liveness Probe

برای اینکه بدانیم اپلیکیشن هنوز زنده ست یا نه اگر Liveness Probe شکست بخورد، فرض می‌شود اپلیکیشن در وضعیت ناپایدار (hang یا crash شده) است و Kubernetes به طور خودکار کانتینر را ریستارت می‌کند.
مناسب برای بررسی ادامه‌ی حیات اپلیکیشن پس از شروع.

Readiness Probe

برای اینکه مشخص شود اپلیکیشن اماده پاسخگویی به درخواست ها هست یا نه اگر Readiness Probe شکست بخورد، Kubernetes پاد را از سرویس‌های Load Balancer و سرویس DNS خارج می‌کند، اما آن را ریستارت نمی‌کند.
مناسب برای کنترل ترافیک به سمت پادها.

Startup Probe

برای بررسی وضیعت اولیه راه اندازی اپلیکیشن استفاده می شود. این Probe برای اپلیکیشن‌هایی مفید است که زمان راه‌اندازی طولانی دارند. تا زمانی که Startup Probe موفق نشود، Liveness و Readiness اجرا نمی‌شوند.
اگر Startup Probe طی زمان مشخصی موفق نشود، پاد ریستارت می‌شود.
مناسب برای جلوگیری از تداخل Liveness با زمان بوت اولیه‌ی طولانی.

سوال هفتم

ReplicaSet

ReplicaSet وظیفه دارد تعداد مشخصی از یک پاد را در کلاستر اجرا و حفظ کند. اگر به هر دلیلی یکی از پادها از بین برود، ReplicaSet به صورت خودکار پاد جایگزین ایجاد می‌کند تا تعداد مشخص‌شده حفظ شود.

با این حال، ReplicaSet به تنهایی ابزار مناسبی برای انجام به‌روزرسانی‌ها نیست، چون خودش نمی‌تواند نسخه‌های جدید از پاد را مدیریت کند.

Deployment

Deployment یک لایه بالاتر از ReplicaSet محسوب می‌شود. زمانی که یک Deployment تعریف می‌کنیم، Kubernetes در پشت‌صحنه یک ReplicaSet برای آن ایجاد می‌کند. اما Deployment امکانات مدیریتی بیشتری مثل:

- Rolling Update
- Rollback
- تاریخچه نسخه‌ها
- مدیریت راحت‌تر با YAML

را در اختیار ما قرار می‌دهد.
بنابراین، Deployment برای استفاده مستقیم توصیه می‌شود، در حالی که ReplicaSet بیشتر در سطح داخلی توسط Deployment استفاده می‌شود و به ندرت مستقیماً ایجاد می‌شود.

DaemonSet

DaemonSet نوع دیگری از آبجکت‌های کنترلی در Kubernetes است که بر خلاف Deployment و ReplicaSet که تعداد مشخصی از پاد را اجرا می‌کنند، وظیفه DaemonSet این است که روی هر نود از کلاستر دقیقاً یک پاد اجرا کند. موارد استفاده DaemonSet:

- اجرای سرویس‌های سیستمی روی همه نودها (مثل Fluentd، Logstash برای جمع‌آوری لاگ)
- اجرای ابزارهای مانیتورینگ مثل Prometheus Node Exporter یا Metricbeat
- اجرای ابزارهای امنیتی یا شبکه‌ای که باید روی همه نودها حضور داشته باشند (مثل CNI pluginها)

DaemonSet تضمین می‌کند که به محض اضافه شدن یک نود جدید به کلاستر، پاد مربوطه نیز به صورت خودکار روی آن اجرا شود.

سوال هشتم

requests

به معنی حداقل منابعی است که یک کانتینر برای اجرا نیاز دارد. وقتی یک پاد ایجاد می‌شود، Kubernetes از این مقدار برای تصمیم‌گیری در مورد اینکه پاد روی کدام نود قرار بگیرد استفاده می‌کند. اگر نودی منابع آزاد کمتر از مقدار requests داشته باشد، پاد روی آن برنامه‌ریزی (schedule) نمی‌شود.
نکته: اگر مصرف واقعی کمتر از requests باشد مشکلی پیش نمی‌آید.

limits

مشخص می‌کند که حداکثر منابعی که یک کانتینر می‌تواند مصرف کند چقدر است. اگر مصرف واقعی کانتینر از این مقدار تجاوز کند:

- برای CPU: مصرف کانتینر محدود می‌شود (throttle).
- برای Memory: کانتینر به احتمال زیاد از بین می‌رود (killed) و پیام خطای OOMKilled ثبت می‌شود.